Incremental Online Verification of Dynamic Cyber-Physical Systems

Lei Bu*, Shaopeng Xing*, Xinyue Ren*, Yang Yang*, Qixin Wang†, and Xuandong Li*
*State Key Laboratory of Novel Software Technology,
Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, P.R.China
†Department of Computing, The Hong Kong Polytechnic University, Hong Kong

***Abstract*—Periodically online verification has been widely recognized as a practical and promising method to handle the non-deterministic and unpredictable behavior of dynamic CPS systems. However, it is a challenge to keep the online verification of CPS systems finishing quickly in time to give enough time for the running system to respond, if any error is detected. Nevertheless, the problems under verification for each cycle are highly similar to each other. Most of the differences are caused by run-time factors like changing of parameters' values or the reorganization of active components in the system. Under this investigation, this paper presents an incremental verification technique for online verification of CPS systems. A method is given to distinguish the differences between the problem under verification and the previous verified problem. Then, by reusing the problem space of the previous verified problem as a warm-start base, the modified part can be introduced into the base, which can be solved incrementally and efficiently. A set of case studies on a real-case train control system is presented in this paper to demonstrate the performance of the incremental online verification technique.**

I. Introduction

By combining communication, computation, and control (3C), Cyber-Physical Systems (CPS) [1] have comprehensive knowledge of their working environment and the other components of the system. Thus, CPS systems can generate accurate instructions, achieve complex targets and gain advantages like reliability and efficiency. However, it is a challenge to keep such kind of complex and dynamic systems working safely.

To assure the safety of the system, modeling and verification of the system's behavior before it is deployed online is a widely used technique. If the formal model is built accurately and passes the verification, then the safety of the system is guaranteed. Formal verification techniques like model checking [2] have been proven successful for many categories of applications, e.g., hardware design. Therefore, it also attracts lots of interest to apply model checking on CPS systems.

Different from other stand-alone devices, the behavior of CPS systems is highly dynamic. Considering a complex system, the structure of the components in the system can be reorganized freely. Considering one component, the values of various control parameters in the system are generated/collected online and updated quickly with high non-determinism. Recall that the basic idea of model checking is to traverse the state space of the behavior model automatically and efficiently to reveal any potential bug. If the complete state space of CPS is not predictable in advance, the system is not verifiable in the design phase in advance.

Online modeling and verification [10], [11], [12], [13], [14] is a promising approach to handle such problems. The basic idea is to take snapshots of the system periodically and frequently. The structure of the current system and the numerical values of all the running parameters will be concretized and kept still in the short-run future. Then, the system behavior in the current cycle is predictable and describable according to the set of the fixed configurations. Therefore, we can get a concrete model according to the current configuration and check the bounded state space of this model as the model only describes the behavior of the system in the given time-bounded short-run future. If any unsafe state is reachable in the bounded state space, the online CPS system will be stopped and switched to an application-dependent fall-back plan immediately to guarantee the safety of the system.

Clearly, the online verification procedure must be as fast as possible to discover bugs before they happen, and to provide enough time for the online systems to react. Due to the existence of both discrete control modes transitions and continuous real-time behavior in CPS systems, hybrid automata [3] is the natural modeling language for CPS systems. As CPS systems have many components involved, the complete model is a network of hybrid automata. Nevertheless, verification of composed hybrid automata is extremely complex. The size of the problem that can be verified offline is rather limited.

To scale the size of the system that the online verification procedure can handle, and to enhance the performance of the verification, we take a re-investigation into the problem under verification. We find the differences between the problems for each cycle mainly come from the following two aspects:

- Most of the non-determinism in the behavior space of CPS systems are caused by the changing of the parameters' values in each cycle. For example, for a train running on a track, the control model for a certain train does not change. The reason we need to verify it again in the next cycle is that many parameters' values are changed, thus the state space is changed accordingly.
- Another aspect which differs the problem space of the system in two consecutive cycles is the continuous reorganization of active components in the system. Still, take a train control system for example, the number of trains running on the track keeps changing. Certain trains

978-3-9819263-2-3/DATE19/©2019 EDAA

can arrive at their destination and leave the track, while several new trains may just depart. When the organization of the active components is changed, clearly the problem space is changed.

Nevertheless, in both of the cases, the major part of the problem space of the two consecutive cycles is the same. For the first case, the model structures are the same for each component. For the second case, the models for the components which are still under operation are the same. In another word, the problems under verification for two adjacent cycles share a large common ground with each other. Therefore, it'll be very useful if we can verify the new problem by reusing the results of the previous verified one as the two problems are similar to each other.

The online verification of each cycle is a typical bounded reachability verification problem [10], [12], [14]. In linear systems, such problems are translated to feasibility problems of a set of linear constraints with respect to the state space of the certain cycle respectively. Then, the feasibility problem can be answered by constraint solving tools like SMT solvers or Linear Programming (LP) solvers. To take advantage of the powerful constraint solving techniques, this paper proposes a method to summarize the differences between the constraint sets with respect to the current cycle and the previous cycle automatically. Then, the corresponding modified linear constraints can be updated on the previous problem space and "incremental linear programming" technique [5] can be deployed to reuse the problem space of the previous solved problem. As a result, the latest problem can be solved efficiently.

To demonstrate the scalability and processing ability of such an incremental verification approach, we conduct a set of case studies on a complex real-case train control CPS system. The experiments show that the performance is optimized substantially. As a result, we can give the running system much more time to conduct the fall-back plan when it is necessary.

II. Background

A. Parametric Hybrid Automata

For a CPS system, the dynamic behavior of the system along with time is a combination of both discrete control modes transitions and continuous real time behavior. Furthermore, the CPS system has intensive interactions with other collaborators in the system and the environment. Many parameters' values in the system are collected online. Thus, parametric hybrid automata [9] is a natural modeling language for CPS.

Definition 1: A parametric hybrid automaton (PHA) is a tuple $H = (X, X^p, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$, where

- X is a finite set of real-valued variables; X^p is a finite set of free parameters; $X \cap X^p = \emptyset$;
- Σ is a finite set of event labels; V is a finite set of *locations*; $V^0 \subseteq V$ is a set of *initial locations*;
- E is a *transition relation* whose elements are of the form $(v, \sigma, \phi, \psi, v')$, where v, v' are in V, $\sigma \in \Sigma$ is a label, ϕ is a set of *transition guards* of the form $f(\vec{y}) \leq a$, and ψ is a set of *reset actions* of the form $x := f(\vec{y})$, where $x \in X$, $a \in \mathbb{R}$, and $y \in X \cup X^p$
- α is a labeling function which maps each location in V to a *location invariant* which is a set of *variable constraints* of the form $f(\vec{y}) \leq a$ where $y \in X \cup X^p$, $a \in \mathbb{R}$.
- β is a labeling function which maps each location in V to a set of *flow conditions* which are of the form $\dot{x} = g(\vec{y})$ where $x \in X$. For any $v \in V$, for any $x \in X$, there is one and only one flow condition $\dot{x} = g(\vec{y}) \in \beta(v)$, where $x \in X$, $y \in X \cup X^p$.
- γ is a labeling function which maps each location in V^0 to a set of *initial conditions* which are of the form $x = a$ where $x \in X$ and $a \in \mathbb{R}$ or $a \in X^p$. For any $v \in V^0$, for any $x \in X$, there is at most one initial condition definition $x = a \in \gamma(v)$.

Given a PHA $H = (X, X^p, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$, For all the location invariants, flow conditions, transition guards and initial conditions in H, if each $f(\vec{y})$ is a linear expression in the form of $\sum_{i=0}^{l} c_i x_i$ and $g(\vec{y})$ is in the form of $[a, b]$, where $x_i \in X$, $c_i, a, b \in \mathbb{R}$ or X^p, we say this PHA is a $PLHA$ (parametric linear hybrid automaton). Furthermore, given a $PLHA$ H, if X^p is an empty set, we say this parametric hybrid automaton is a classical concrete linear hybrid automaton (LHA) as defined in [3].

B. Scenario-based Online Modeling and Verification

As the main idea of model checking is traversing the complete state space of the system to find the bug, offline model checking of dynamic CPS is infeasible due to the unpredictable configurations. To address this problem, a natural strategy is to carry out online model checking instead. The basic idea is to periodically sense/collect the related CPS configurations during run-time, and concretize the PHAs into conventional hybrid automata. We then carry out bounded model checking of the updated model to predict if the CPS system can reach any unsafe states in the short-run future. If so, an alarm is raised to trigger an application-dependent fall-back plan.

One problem that the designers are very interested in is the safety-critical scenario validation problem, which asks whether the system can reach certain unsafe states via a certain sequence of actions. That is, if certain "*scenario*" can happen [9], [15]. Typically, a scenario can be projected to each component as a sequence of control modes/locations in the structure of the model. Thus, the verification of the existence of a certain scenario can be transformed to the reachability problem of a path set which is composed of one path from each automaton [9].

Such composed-path reachability verification of LHA has been studied intensively. Study [4] presented a typical approach for such problems. It checks a group of paths at a time, one path for each LHA. The reachability problem along those specific paths can be reduced to a linear program and be solved by LP easily. First, all of the paths are transformed into a group of linear constraints automatically. Then, a few constraints about the system integration according to the synchronization events in each path will be added to ensure that the components cooperate correctly.

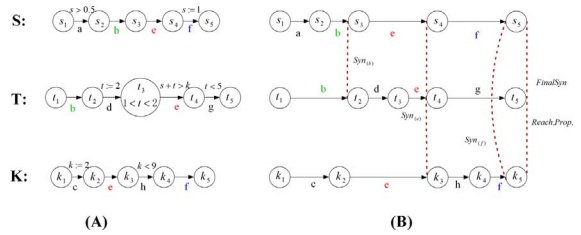

Fig. 1. Sample Scenario And The Scenario-based Reachability Encoding

For example, Fig.1.A gives a simple scenario consisting of three paths from different components: S, T, and K which synchronize with each other by shared labels b, e, and f. Each system has one variable, s for S, t for T, k for K. The flow conditions for all the variable are unified as $\dot{\vec{x}} \in [0.9, 1.1]$ in all the locations. The specification is whether the property $s + 2t - 3k = 0$ can be satisfied at the global location (s_5, t_5, k_5).

In [4], a group of linear constraints is generated for these three paths. The constraints encode the potential timed behavior corresponding to the paths. Take the path $\langle t_1 \rangle \rightarrow \langle t_2 \rangle \rightarrow \langle t_3 \rangle \rightarrow \langle t_4 \rangle \rightarrow \langle t_5 \rangle$ in T for example, we can use $\left\langle \begin{array}{c} t_i \\ \delta_i^t \end{array} \right\rangle$ to indicate that the system has stayed in location t_i for time delay δ_i. The behavior of the system is represented by $\left\langle \begin{array}{c} t_1 \\ \delta_1^t \end{array} \right\rangle \rightarrow \left\langle \begin{array}{c} t_2 \\ \delta_2^t \end{array} \right\rangle \rightarrow \left\langle \begin{array}{c} t_3 \\ \delta_3^t \end{array} \right\rangle \rightarrow \left\langle \begin{array}{c} t_4 \\ \delta_4^t \end{array} \right\rangle \rightarrow \left\langle \begin{array}{c} t_5 \\ \delta_5^t \end{array} \right\rangle$.

For each location t_i, two variables $\lambda_i(t)$ and $\zeta_i(t)$ are generated to represent the valuation of t when entering t_i and leaving t_i after staying there by δ_i time units.

First, the time constraints enforced by the local system T must be satisfied, which forms a group of linear constraints about $\lambda_i(t)$, $\zeta_i(t)$ and δ_i^t. Take the location t_3 for example:

1.1: According to the flow condition, $1.1\delta_3^t + \lambda_3(t) \geq \zeta_3(t) \geq 0.9\delta_3^t + \lambda_3(t)$.

1.2: For the invariant $1 < t < 2$, we have $1 < \zeta_3(t) < 2$ and $1 < \lambda_3(t) < 2$.

1.3: For the transition guard $t < 5$ on the local transition g, we have $\zeta_4(t) < 5$.

1.4: For the reset action $t = 2$ on the local transition d, we have $\lambda_3(t) = 2$.

Besides the local constraints for each component, synchronization constraints will be added to ensure these components cooperate accurately w.r.t. the synchronization events, which are illustrated by $SYN_{(event)}$ in Fig.1.B.

2.1: For the event b shared by S and T, $\delta_1^t = \delta_1^s + \delta_2^s$.

2.2: For the variable communication in transition, e.g., $s+t > k$ in e, we have $\zeta_3(s) + \zeta_3(t) > \zeta_2(k)$.

2.3: All the components have spent exactly the same time, e.g., for S and T, we have $\delta_1^s + \delta_2^s + \delta_3^s + \delta_4^s + \delta_5^s = \delta_1^t + \delta_2^t + \delta_3^t + \delta_4^t + \delta_5^t$.

2.4: For reachability specification $s + 2t - 3k = 0$, we get $\zeta_5(s) + 2\zeta_5(t) - 3\zeta_5(k) = 0$.

Above all, the scenario-based reachability analysis problem is transformed into a feasibility problem of a set of linear constraints. It is well-known that the feasibility problem of linear constraints can be solved by LP technique efficiently.

III. Incremental Verification

In the last section, we give a quick review of the online modeling and verification framework presented for CPS systems. By taking the snapshot and freezing the dynamic run-time aspects, a concrete static model of the system's time-bounded behavior in the short-run future is describable and verifiable. Then scenario-based reachability verification is deployed to check whether the certain dangerous scenario will happen under the current configuration.

As the verification is conducted online, we have to finish the verification as quickly as possible. If we cannot answer the verification questions before the system configuration is updated or the error happens, the result will be useless. Furthermore, if a risk is reported by the verification, we need to give the system enough time to respond. Otherwise, the system can not avoid the dangerous error.

Basically, the online verification is checking the same problem on "different" systems periodically. The "same problem" under verification here is the existence of certain bad scenarios, which do not change, while the "differences" between the system models in different snapshots are lightweight.

Recall that, the motivation of CPS has to be verified online is that the running configuration of the system like parameters' values, organizations of active components and etc., are changing frequently, which will cause the state space of the system behavior to be changed correspondingly.

Despite whether the values of the parameters and/or the organization of active components are changed, the major part of the problem space under verification for two nearby cycles are the same. For parameters update, the model structure keeps the same for each component. For the reorganization of active components, the parametric models for the components which are still active are not changed. In another word, the problems under verification share a large portion of common ground with each other. Therefore, it'll be very useful if we can verify the new problem by reusing the result of the previous verified one instead of starting from scratch.

Formally speaking, we build the static time-bounded model M_1 for cycle 1, verify M_1 with respect to property p, and get $M_1 \vDash p$. In cycle 2, either parameters and/or the organization of active components in the system are updated. The corresponding model for cycle 2 is M_2. Now, we need to verify whether $M_2 \vDash p$ holds. Clearly, we can check this property as an independent task. However, M_1 and M_2 are similar with each other. We mark the differences between M_2 and M_1 is $M_2 \ominus M_1 = \Delta m$. Can we reuse the verification effort devoted in checking $M_1 \vDash p$ to accelerate the checking of whether $M_1 \oplus \Delta m \vDash p$?

As reviewed in Sect.II.B, the scenario-based reachability problem is encoded into the feasibility problem of a set of linear constraints, which can be solved by LP technique efficiently. We mark the linear constraint set corresponding to whether $M_1 \vDash p$ as ρ_1, and the constraint set for $M_2 \vDash p$ as ρ_2. What we want is to accelerate the solving of ρ_2 by taking advantage of solving ρ_1.

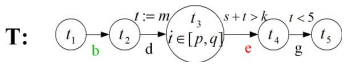

Fig. 2. Sample Parametric HA T

Luckily, incremental solving is a powerful mechanism supported in LP. When solving an LP problem, the structure like "basis" is built for the problem space. This "basis" constitutes a starting point for the solution of a nearby optimization problem to save computational effort. Many techniques are available in this context, like advanced basis [6], [7], warm-start strategies [5] and etc, which are named as "Incremental Linear Programming" in general. This method can significantly reduce the number of iterations and the computation time. Most of the optimization tools have the ability to reuse the basis of the original problem.

Therefore, as a user of LP technique, the problem we need to concern here is how to locate the $\Delta\rho = \rho_2 \ominus \rho_1$, which is the set of modified linear constraints, including the set of constraints added, deleted and changed on ρ_1, caused by system dynamics like parameter update and system reorganization. Then, incremental LP techniques can be applied to deploy $\Delta\rho$ to the problem basis constructed when solving ρ_1, and reuse the basis to accelerate the solving of the feasibility problem of ρ_2 which is corresponding to whether $M_2 \vDash p$.

In the following, we describe how to generate the differential constraint set $\Delta\rho$ from the two main directions: Parameter Update and System Reorganization:

Parameter Update: During a system is in operation, the values of the control parameters used in the system are updated frequently, which will affect the system behavior accordingly. Recall the LP encoding mechanism described in Sect.II.B, the updating of the parameters' values will only affect the related constraints in the local linear constraint set of the component.

For example, Fig.2 is a parametric version of model T in Fig.1.A, that the reset action on transition d is modified to $t := m$ from $t := 2$, where m is a free parameter. Furthermore, the flow condition on location t_3 is modified to $\dot{t} \in [p, q]$, where p and q are free parameters.

The corresponding constraints in items 1.1 and 1.4 from Sect.II.B are translated to $q\delta_3^t + \lambda_3(t) \geq \zeta_3(t) \geq p\delta_3^t + \lambda_3(t)$, and $\lambda_3(t) = m$ respectively, where p, q, m are free parameters, and their values can be updated in each cycle according to the run-time numeric values.

System Reorganization: Except for parameter update, the dynamic behavior of the CPS system also comes from the reorganization of active components in the system. For example, for a railway track, the number of trains running on the track is changing all the time. Another example is industrial IoT, machines can reorganize with respect to different tasks on demand. For systems like these dynamic CPS systems, the organization of the active components in each cycle keeps changing. Therefore, the problem model under verification will be changed accordingly.

The generation of differential constraint set $\Delta\rho$ for system reorganization is more difficult than parameter update which only needs to update the coefficients' values in the constraint set. For system reorganization, clearly, the constraints of the components left the system have to be deleted from the constraint set and the constraints of the components which just join the system will be added. Besides, we have to deal with the constraints about the synchronization among components. Now, we still use the scenario given in Fig.1 for example to illustrate the generation of the $\Delta\rho$.

The system given in Fig.1 has three components S,T and K included. Suppose in the next cycle, T lefts the system, while a new system L joins the system. To show this in Fig.3.A graphically, T is covered by a grey ellipse with shadow and L is surrounded by a red rectangle. The corresponding modification of the synchronization-related linear constraints are shown in Fig.3.B as follows:

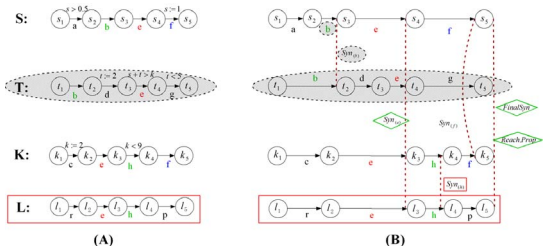

Fig. 3. System Reorganization Related Encoding Modification Demonstration

- Shared labels will be reanalyzed.
  - Shared label b between S and T is a local event now, as T is removed. As a result, the corresponding synchronization constraint Syn_b, equation 2.1 in Sect.II.B, is deleted.
  - Local label h in K becomes a shared label in the new system, as it is shared by K and L. Therefore, the corresponding constraint Syn_h will be added.
  - Shared label e of S, T, and K are now shared by S, K, and L. Therefore Syn_e will be updated.
- Final synchronization constraint will be updated.
  - As components in the system are changed, the final synchronization constraint will be updated as we have to make sure the total time spent by L must equal with other components.
  - The encoding of the reachability property will be updated too, if the new property is related to any changed component.

Above all, by combing the constraints modified in the stage of both aspects, we can generate the differential constraint set $\Delta\rho = \rho_2 \ominus \rho_1$ between the models of the system in two adjacent cycles. By introducing $\Delta\rho$ into the "advanced basis" of the LP procedure for solving ρ_1, the same LP procedure can be "warm-started" quickly and solve ρ_2 much more efficiently.

IV. Case Studies

In order to illustrate the feasibility of the incremental verification technique presented in this paper, we conduct a set of case studies on a state-of-the-art train control CPS system.

A. System Description

The system is a communication-based train control system (CBTC), which is the state-of-the-art train control system and a typical CPS system introduced in [10]. The parametric model of this CBTC system is given in Fig.4. The CBTC system communicates with the control center frequently, twice per second, to get the latest movement authority (MA) which indicates the place that the train is permitted to go before receiving a new command. Then the CBTC system will compute the new velocity curve for the next operation cycle autonomously by taking account of the MA it received and the current operation status of the train like weather condition, track condition, train mass and etc. Clearly, these parameters' run-time values are highly random and these values are difficult, even impossible, to predict offline.

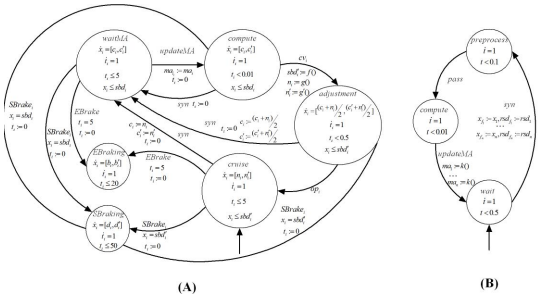

Fig. 4. PHA Model For $Train_i$ and Control Center

During a CBTC system is in operation, there are several safety rules that the system must obey. For example, as the CBTC system relies on communication, once the communication channel is broken and trains have not received MA update from the control center for 5 seconds, all the trains will brake emergently to avoid colliding with each other. The scenario which is under verification is when the communication corrupts, whether train $Train_i$ will collide with the train ahead $Train_{i-1}$. The complete composed system to verify is a network consists of models from $Train_1$ to $Train_n$. The reachability specification is whether the physical location of $Train_i$ will equal with $Train_{i-1}$.

B. Experimental Data

The incremental verification mechanism presented in this paper is implemented in BACH$_{OL}$ [16] which supports online model checking of PHA models. The reason we use BACH$_{OL}$ is that it is a typical LP based checker designed for online verification of CPS. The LP solver under BACH$_{OL}$ is IBM CPLEX [8] which supports "Incremental LP" by "advanced basis" technique [6], [7]. The experiments are conducted on a Think Center (Windows 7 Professional 64 bit, Intel Core2 Quad CPU $3.1\text{GHz}, 4\text{GB}$ RAM).

To demonstrate the processing capacity to handle large dynamic CPS systems by incremental verification, we deploy the technique on a railway system which has 100 trains running on it. We generate a script which includes the detailed information for each cycle, e.g. the list of active trains in that cycle and the parameters' readings of the certain train. The script has data of 1000 cycles included. As the communication period of the CBTC system is 500 ms, the script describes the system behavior in 500 seconds.

The first experiment considers parameter update. In another word, none of the trains join or quit the track during operation. The statistic data is shown below in Table.I. The time spent on model building and LP problem construction for each cycle is marked as "Build", the time spent for the LP solving is marked as "Solve" and the time for the complete workflow is marked as "Total". We report the data for the non-incremental way which builds a new problem from scratch then solve, marked as "Non-Incre." and the incremental way as "Incre.". We also report the reduction ratio ("Speedup") and standard deviation ("Std."). From these data, we can see that by reusing the previous problem basis, and check the new model incrementally, the time spent for each step in the cycle is reduced significantly. For example, the mean time for problem construction is deduced from 158.98 ms to 84.88 ms, that 46.6% of the time is saved. While for LP solving, the time is deduced from 20.52 to 4.81, which means 85.6% of the time was saved. In total, half of the time is saved.

TABLE I
STATISTICS DATA ON 1000 CYCLE TRACE OF PARAMETER UPDATE

Tech.	Build		Solve		Total	
	Mean(ms)	Std.	Mean(ms)	Std.	Mean(ms)	Std.
Non-Incre.	158.98	33.44	20.52	2.15	179.5	33.4
Incre.	84.88	3.34	4.81	1.41	89.6	4.3
Speedup	46.6%		85.6%		50.1%	

In the above experiment, the structure of the system remains stable. In real cases, the organization of the system is dynamic as several trains may quit as they have reached their destinations and some other trains will join the system as they just depart from the station. Therefore, the second experiment we conduct includes both parameter update and system reorganization. First, each train may enter and leave the system randomly. Meanwhile, if a train is onboard, the running parameters keep changing. The new trace data also covers 1000 cycles. The number of trains for each cycle fluctuates around 100 in the range of [90,110]. The statistical data is shown in Table.II.

TABLE II
STATISTICS DATA ON 1000 CYCLE TRACE OF PARAMETER UPDATE AND SYSTEM REORGANIZATION

Tech.	Build		Solve		Total	
	Mean(ms)	Std.	Mean(ms)	Std.	Mean(ms)	Std.
Non-Incre.	160.95	36.73	19.58	2.15	180.5	17.3
Incre.	112.71	18.19	9.12	2.75	121.8	19.5
Speedup	30%		53.4%		32.5%	

In the data, we can see by introducing both parameter update and system reorganization into the system, the modification of the model is more complicated than just updating parameters' values. $BACH_{OL}$ needs to spend more time distinguishing the differences between the new model and the previous one

to generate the differential constraint set. Therefore, the time spent on modeling and LP construction in "Incre." is raised from 84.88 ms to 112.71 ms. Nevertheless, still 30% smaller than the "Non-Incre." value 160.95 ms. Consequently, the mean time that the incremental checking technique costs is 9.12 ms which is larger than 4.81 ms in the case with only parameter update, but the reduction ratio is still more than 50%, which is remarkable. In general, the mean total time for the checking of the system in each cycle is 121.8 ms, which is 32.8% smaller than the non-incremental solving.

V. Related Work

Verification of Parametric Real-time System. The PHA studied in this paper is an extension of parametric timed automata (PTA) [20] and slope parametric linear hybrid automata (SPLHA) [19]. Existing verification studies of PTA, SPLHA, and related variants mainly focused on the parameter synthesis to make the system satisfy the desired property. Meanwhile, there are also other properties of PTA studied, including reachability, unavoidability and so on. However, most of the non-trivial problems on PTA are proven to be undecidable [21]. Differently, our work does not focus on parameter synthesis, we conduct online verification to see whether the run-time control parameters are safe or not in the short-run future.

Incremental Verification of Parametric System. In the era of CPS, it is common to see systems working in dynamic environments. Therefore, the verification of parametric system is an emerging topic recently [22]. Similar to the idea presented in this work, studies [18], [23] explore the similarity between instances of parameterized systems to re-use computations for different instantiations and perform state-elimination for verification of parametric Markov chains.

In this paper, the incremental online verification of safety-critical scenario is achieved by taking advantage of incremental LP solving. Study [17] shares a similar motivation and proposes an incremental mechanism for QBF solving. They applied this method in the incremental verification of partial designs [24]. This technique can be applied directly into the BMC-based online verification using the methodology given in this paper in the future.

VI. Conclusion

Online verification of dynamic CPS system is a well-recognized method which has attracted a lot of attention recently. However, in order to handle real-case CPS systems, we need to raise the efficiency of the online verification framework.In this paper, we present an incremental verification method to distinguish the differences between the system models in two adjacent cycles, and generate the differential constraint set automatically. By taking advantage of incremental linear programming technique, the verification of the new model can start from the "shoulder" of the previous cycle to save time and raise the efficiency. We conduct a set of case studies on a train control system which has around 100 trains online. The experiment data shows by using our incremental verification method, the efficiency is optimized significantly.

Acknowledgment

The authors in Nanjing University are supported by the National Key Research and Development Plan (No. 2017YFA0700604), and the National Natural Science Foundation of China (No.61632015, 61561146394, No.61572249). The author in Hong Kong Polytechnic University (HK PolyU) is supported by Hong Kong RGC GRF PolyU152164/14E, RGC ECS PolyU5328/12E, RGC Germany/HK Joint Research Scheme G-PolyU503/16, and The Hong Kong Polytechnic University fund G-YN37, G-UA7L, G-YBMW, G-YBXW, 1-BBWC, and 4-ZZHD.

References

[1] LEE, E. 2006. Cyber-physical systems - are computing foundations adequate? Position paper for NSF workshop on Cyber-Physical Systems: Research Motivation, Techniques and Roadmap.
[2] CLARKE, E., *et al*.. 1999. Model Checking. MIT Press.
[3] HENZINGER, T. 1996. The theory of hybrid automata. In Proc. of LICS'96, 278-292.
[4] BU, L. AND LI, X. 2011. Path-oriented bounded reachability analysis of composed linear hybrid systems. STTT. 13:4, 307-317.
[5] JOHN, E., *et al*. 2008. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimensions. In Comput. Optim., Appl. 41,151-183.
[6] I. MAROS, G. MITRA. 1998. Strategies for creating advanced bases for large-scale linear programming problems. In INFORMS Journal on Computing, 10(2), 248-260.
[7] R.E. BIXBY. 1992. Implementing the Simplex Method: The Initial Basis. In ORSA Journal on Computing, 4:3, 267-284.
[8] CPLEX. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
[9] BU, L., *et al*. Technical Report, 2018. Scenario Reachability Validation based Online CPS Fault Prediction.
[10] BU, L., *et al*. 2011. Toward online hybrid systems model checking of cyberphysical systems time-bounded short-run behavior. In ACM SIGBED Review, 8(2): 7-10.
[11] LI, T., *et al*. 2012. From Offline toward Real-Time: A Hybrid Systems Model Checking and CPS Co-Design Approach for Medical Device Plug-and-Play (MDPnP). In Proceedings of ICCPS2012, 13-22, IEEE.
[12] STANLEY, B., *et al*, 2014, Real-time reachability for verified simplex design. In Proceedings of RTSS 2014,138–148, IEEE.
[13] NGUYEN, L., *et al*. 2015. Runtime Verification for Hybrid Analysis Tools. In Proceedings of RV 2015, 281–286, IEEE.
[14] CHEN, X., *et al*. 2017. Model Predictive Real-Time Monitoring of Linear Systems. In Proceedings of RTSS 2017, 297–306, IEEE.
[15] NAJM, W., *et al*. 2007. Pre-crash scenario typology for crash avoidance research, In DOT HS.
[16] BU, L., *et al*. 2012. Demo Abstract: BACH$_{OL}$ - Modeling and Verification of Cyber-Physical Systems Online. In ICCPS2012, pp.222, IEEE.
[17] MARTIN, P., *et al*. 2012. Verification of Partial Designs Using Incremental QBF Solving, In Proceedings of DATE 2012, 623-628, IEEE.
[18] GAINER, P., *et al*. 2018. Incremental Verification of Parametric and Reconfigurable Markov Chains. In QEST 2018, 140-156, Springer.
[19] ADÉLAïDE, M., *et al*. 2002. A Class of Decidable. In Proceedings of AMAST 2002, 132–146.
[20] ALUR, R., *et al*. 1993. Parametric real-time reasoning. In Proceedings of the 25th Annual ACM Symposium on Theory of Computing, 592–601.
[21] ANDRÉ, E., 2015, What's Decidable About Parametric Timed Automata?, In Proceedings of FTSCS 2015, 52–68
[22] ISENBURG, T. 2017. Incremental Inductive Verification of Parameterized Timed Systems. ACM TECS. 16(2): 47:1-47:24.
[23] KWIATKOWSKA, M., *et al*. 2011. Incremental quantitative verification for markov decision processes. In Proceedings of International Conference on Dependable Systems & Networks. 359370. IEEE.
[24] MILLER, C., *et al*. 2015. Verification of partial designs using incremental QBF. AI Commun. 28(2): 283-307.